OS Project : Memory Management

内存管理之请求调页管理方式

项目分析

项目描述

假设每个页面可存放10条指令，分配给一个作业的内存块为4。模拟一个作业的执行过程，该作业有320条指令，即它的地址空间为32页，目前所有页还没有调入内存。

给定条件下简单的模拟一个内存调度过程, 置换算法使用FIFO算法或LRU算法

开发环境

- 开发工具: VS Code + Chrome
- 开发语言: html+css+js

需求分析

- 分配给该作业4个内存块, 每个页面可以存放10条指令,该作业有320条指令(也就是占32页)
- 随机生成320条指令的序列, 50%顺序执行, 25%均匀分布在前地址, 25%均匀分布在后地址
- 对于生成的指令序列进行320次模拟执行
- 如果所访问指令在内存中, 显示其物理地址, 并转到下一条指令
- 如果发生缺页, 则记录缺页次数, 并调入内存
- 如果四个内存块已经占满, 则分别使用FIFO和LRU算法进行置换
- 显示最后的缺页率

实现方法

随机指令生成

1. 在$0 \rightarrow 319$条指令中, 随机出一个起始指令, 记作m, 执行m
2. 选择$m+1$作为下一条顺序执行的指令
3. 用随机函数在$0 \rightarrow m-1$中随机一个数作为前段指令, 将其记作m_1
4. 选择m_1+1作为下一条顺序执行的指令
5. 用随机函数在$m_1+2 \rightarrow 319$中随机一个数作为前段指令, 将其记作m_2
6. 选择m_2+1作为下一条顺序执行的指令
7. 循环执行步骤3~6, 直至运行320次为止,以期望得到 "50%顺序执行, 25%均匀分布在前地址, 25%均匀分布在后地址"的指令序列

算法实现

FIFO(First In First Out)先进先出算法

FIFO 算法是最简单的页面置换算法。FIFO 页面置换算法为每个页面记录了调到内存的时间，当必须置换页面时会选择最旧的页面。

FIFO 页面置换算法的优点是易于理解和编程, 然而 缺点也很明显,FIFO的效率并不理想.FIFO只记录页面第一次进入内存的时间,因此如果第一次进入的页面是常用页面, 那么就会发生多次没意义的缺页中断, 降低性能.

建一个 FIFO 队列，来管理所有的内存页面。置换的是队列的首个页面。当需要调入页面到内存时，就将它加到队列的尾部。

```
var page = Math.floor(ins / 10);
var findRes = findPage(page, memory);
var tag = false;
//命中
if(findRes >= 0){
    memoryBlock = findRes + 1;
    bIsMissing = false;
}

//不命中
else{
    bIsMissing = true;
    missingPages++;
    var length = memory.length;
    //调入
    if(length < 4){
        memory[length] = page;
        memoryBlock = length + 1;
        fifoQueue[length] = page;
    }
    //替换
    else{
        tag = true;
        var pageReplaced = fifoQueue[0];
        var memReplaced = findPage(pageReplaced, memory);
        memory[memReplaced] = page;
        memoryBlock = memReplaced + 1;
        for(var i = 0; i < 3; i++){
            fifoQueue[i] = fifoQueue[i + 1];
        }
        fifoQueue[3] = page;
    }
}
```

LRU(Least Recently Used)缓存淘汰策略

LRU算法是最近最久未被使用的一种置换算法。利用局部性原理，根据一个作业在执行过程中过去的页面访问历史来推测未来的行为。它认为过去一段时间里不曾被访问过的页面，在最近的将来可能也不会再被访问。所以，这种算法的实质是：当需要淘汰一个页面时，总是选择在最近一段时间内最久不用的页面予以淘汰。

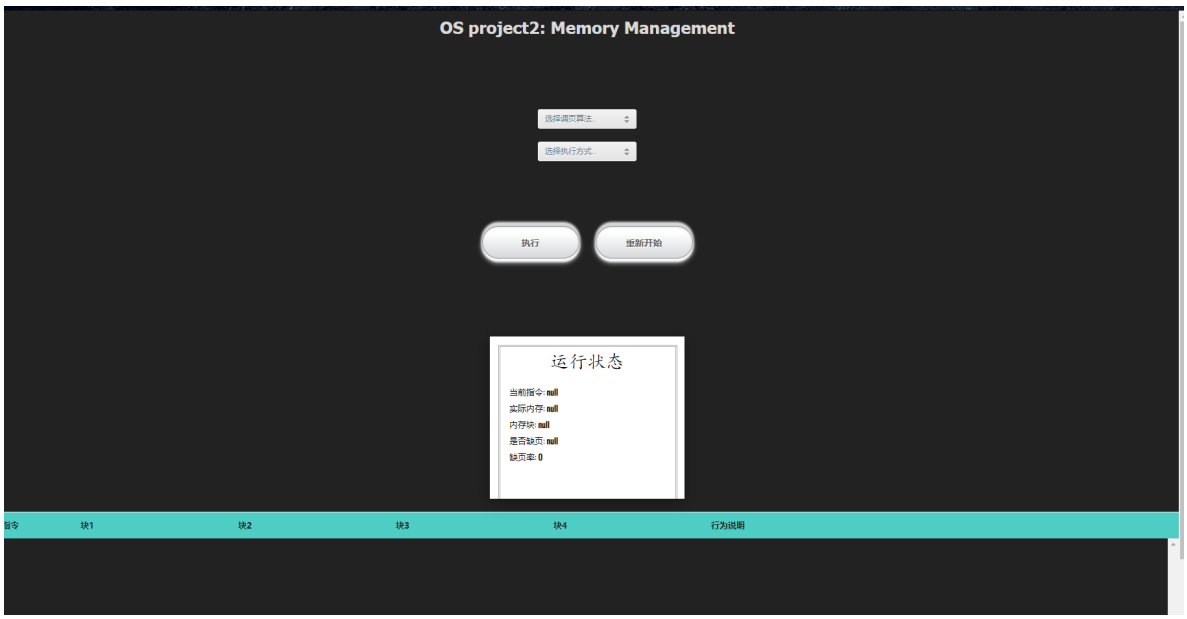
根据局部性原理, LRU算法是最接近最佳适配算法的调换算法, 性能较好, 但是对硬件开销比较大

因为本项目中分配的内存块只有4, 考虑到缺页率一般小于50%, 故实现中的LRU数组存储的是4个内存块上一次被使用的时间, 每当缺页时遍历查找最久未被使用的内存块进行调换

```
function LRU(ins){
    var page = Math.floor(ins / 10);
    var findRes = findPage(page, memory);
    var tag = false;
    //命中
    if(findRes >= 0){
        memoryBlock = findRes + 1;
        bIsMissing = false;
        lruQueue[findRes] = runTimesCnt;
    }

    //不命中
    else{
        bIsMissing = true;
        missingPages++;
        var length = memory.length;
        //调入
        if(length < 4){
            memory[length] = page;
            memoryBlock = length + 1;
            lruQueue[length] = runTimesCnt;
        }
        //替换
        else{
            tag = true;
            var leastInsIndex, leastIns = 100000;
            for(var i = 0; i < 4; i++){
                if(lruQueue[i] < leastIns){
                    leastIns = lruQueue[i];
                    leastInsIndex = i;
                }
            }
            memory[leastInsIndex] = page;
            memoryBlock = leastInsIndex + 1;
            lruQueue[leastInsIndex] = runTimesCnt;
        }
```

成果展示



指令	块1	块2	块3	块4	行为说明
176	17	Empty	Empty	Empty	缺页调入: 将第17页调入第1块内存
177	17	Empty	Empty	Empty	命中: 177在第1块内存中
279	17	27	Empty	Empty	缺页调入: 将第27页调入第2块内存
280	17	27	28	Empty	缺页调入: 将第28页调入第3块内存

令	块1	块2	块3	块4	行为说明
88	18	Empty	Empty	Empty	缺页调入: 将第18页调入第1块内存
89	18	Empty	Empty	Empty	命中: 189在第1块内存中
74	18	27	Empty	Empty	缺页调入: 将第27页调入第2块内存
75	18	27	Empty	Empty	命中: 275在第2块内存中
25	18	27	22	Empty	缺页调入: 将第22页调入第3块内存
26	18	27	22	Empty	命中: 226在第3块内存中